**Project title:** Multi-Owner data Sharing for Analytics and Integration respecting Confidentiality and OWNer control
**Project acronym:** MOSAICrOWN
**Funding scheme:** H2020-ICT-2018-2
**Topic:** ICT-13-2018-2019
**Project duration:** January 2019 – December 2021

# D5.1

# First Version of Data Sanitisation Tools

Editors: Enrico Bacis (UNIBG)
Dario Facchinetti (UNIBG)
Stefano Paraboschi (UNIBG)
Matthew Rossi (UNIBG)
Reviewers: Megan Wolf (MC)
Rigo Wenning (W3C)

**Abstract**

The deliverable presents the tools developed within MOSAICrOWN to manage sanitization. The first tool anonymizes a collection of sensitive data, applying $k$-anonymity. This tool adapts the Mondrian algorithm to the Apache Spark framework, which permits to use the computational capacity of a distributed system. The second tool supports the verification of the exposure of a trained machine learning model to the membership inference attack. This supports the sanitization activity, providing a verification that a model built from sensitive data does not leak information that the user expects to be protected.

| Type | Identifier | Dissemination | Date |
|---|---|---|---|
| Deliverable | D5.1 | Public | 2020.05.31 |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825333.

# MOSAICrOWN Consortium

| | | | |
|---|---|---|---|
| 1. | Università degli Studi di Milano | UNIMI | Italy |
| 2. | EMC Information Systems International | EISI | Ireland |
| 3. | Mastercard Europe | MC | Belgium |
| 4. | SAP SE | SAP SE | Germany |
| 5. | Università degli Studi di Bergamo | UNIBG | Italy |
| 6. | GEIE ERCIM (Host of the W3C) | W3C | France |

# Versions

| Version | Date | Description |
|---|---|---|
| 0.1 | 2020.05.05 | Initial Release |
| 0.2 | 2020.05.26 | Second Release |
| 1.0 | 2020.05.31 | Final Release |

# List of Contributors

This document contains contributions from different MOSAICrOWN partners. Contributors for the chapters of this deliverable are presented in the following table.

| Chapter | Author(s) |
|---|---|
| Executive Summary | Stefano Paraboschi (UNIBG) |
| Chapter 1: Introduction | Enrico Bacis (UNIBG), Dario Facchinetti (UNIBG), Stefano Paraboschi (UNIBG), Matthew Rossi (UNIBG) |
| Chapter 2: Sanitization in Apache Spark | Enrico Bacis (UNIBG), Sabrina De Capitani di Vimercati (UNIMI), Dario Facchinetti (UNIBG), Sara Foresti (UNIMI), Luca Ghislotti (UNIBG), Giovanni Livraga (UNIMI), Stefano Paraboschi (UNIBG), Matthew Rossi (UNIBG), Pierangela Samarati (UNIMI) |
| Chapter 3: Support for Differential Privacy | Jonas Boehler (SAP SE), Daniel Bernau (SAP SE) |
| Chapter 4: Conclusions | Stefano Paraboschi (UNIBG) |

# Contents

# List of Figures

# Executive Summary

The deliverable reports on two of the tools developed within MOSAICrOWN to implement sanitization in digital data markets. The first tool implements an algorithm for protecting of sensitive data based on $k$-anonymity. The tool operates within Apache Spark and supports a distributed computation of the anonymization. The integration with one of the most used frameworks for executing computations on big data aims at the management of scalability. The second tool supports the verification that a prediction model built from sensitive data is not exposed to the membership inference attack, which allows an adversary to determine whether a given record was present in the data set used for the construction of the model. An important strategy for sanitization consists in building a synthetic prediction model from a collection of data. It is important to have a tool that verifies whether the release of a sanitized model can indeed be used to infer on the presence of data items in the training data set. If no check is applied, the prediction model will often present an overfit on the training data.

The deliverable is produced at M17, near the end of the first period of the project, before the first version of the MOSAICrOWN Policy Language has been finalized. It is planned that the tool for the anonymization will eventually be integrated with the policy language, supporting the application of anonymization based on the requirements expressed in the policy.

The tools described in this deliverable are not the only effort toward sanitization in MOSAICrOWN. Use Case 3 (UC3) is focused on the support of advanced sanitization techniques and, near the end of the project, the deliverables produced by Work Package 2 will report on the results produced in the construction of the UC3 environment. The tools that have been selected are the tool for $k$-anonymization and the tool that considers the protection from membership inference attacks, currently receiving limited support in machine learning platforms. The work toward the refinement and extension of the tools functionality will continue, with a release of the final version of the tools planned for Deliverable D5.4 “Final version of tools for data sanitisation and computation”, at M33 (September 2021).

The deliverable consists of 4 chapters. Chapter 1 introduces the scenario where the tools are going to be applied and the motivation for their design. Chapter 2 describes the tool for the sanitization of large data collections based on $k$-anonymity operating over Spark. The chapter first describes the main features of the Apache Hadoop and Apache Spark frameworks.

It then describes the main features of the Mondrian anonymization algorithm. Mondrian’s approach to enforce $k$-anonymity leverages classical algorithms for the construction of multidimensional indexes integrating them with metrics for the evaluation of the information loss produced by a given organization of the tuples in buckets of at least size $k$. Then, the main features of the implemented tool are described and a perspective is presented on the integration of the tool with the MOSAICrOWN Policy Language. Chapter 3 describes the motivation and the architecture of the tool for the management of membership inference attacks. Finally, Chapter 4 illustrates some concluding remarks, discussing the impact that the work on sanitization has in the construction of future digital data markets.

# 1. Introduction

MOSAICrOWN provides support for the implementation of data protection requirements and can combine those with protection techniques for a variety of data formats. The tools implementing the protection techniques have to be innovative with respect to several aspects. They need to integrate well into the architecture of MOSAICrOWN. But they also need to support the declarative representation of policies attached to data and taken into account at processing time to reflect directives often, but not exclusively, deriving from preferences given by data subjects. The heterogeneity of the project Use Cases testifies of the ambition of the project in proposing solutions that can be adapted to a variety of scenarios, in terms of application domains, technologies, and profile of the expected users.

The MOSAICrOWN Policy Language is currently being designed and is going to be described in Deliverable D3.3. The policy language will support the representation in a formal way of the requirements for the protection of data, looking at two families of techniques: (a) data wrapping (investigated in Work Package 4) is based on the use of cryptography and other transformations that encapsulate the information content, with the ability to go back to the original values of the data; and (b) sanitization (investigated in Work Package 5) is based on a number of solutions that modify at fine granularity the information content of the data, reducing its precision by omitting portions of the information or adding some noise; the reduction in precision produced by sanitization is calibrated, to balance the protection of sensitive information and at the same time maximizing utility in the access to the sanitized data. The current status of the tools developed for data wrapping are described in Deliverable D4.1. This deliverable describes the status of the tools for the application of sanitization.

The integration between the MOSAICrOWN Policy Language and the tools implementing the protection techniques is an important property. The work on the policy language is still ongoing and results on this direction are planned for the end of the project. The tools has been designed taking into account this requirement and no obstacles are expected toward a successful integration. The availability of software tools implementing the protection techniques will support a formalization able to capture in a correct way the representation of the requirements of this domain. The core of the effort in this direction will occur in the second half of the project.

The objectives of the construction of tools in MOSAICrOWN is both the scientific investigation of novel approaches and techniques, and the future support to concrete industrial use cases, which will see the deployment of solutions that are representative of the state of the art in data protection. The role of tools in these innovation projects is to support the testing of ideas, facilitating the transfer of novel approaches to industrial applications and practical scenarios. Tools are crucial to enhance the cooperation between research and industrial partners.

It is to note that the implementation effort within MOSAICrOWN toward the construction of tools for data sanitization is only partially represented by this deliverable. There is a significant development effort that is dedicated to the realization of the support of sanitization within the use cases. There are also investigations on scientific topics that are supported by implementation

efforts. The tools that are being presented are those that have been considered the most effective in illustrating, at this point in the project, the support of the sanitization goals.

As it is common for software tools, the development work is continuous and will go on after the preparation of this deliverable, with a sequence of advancements that will be produced in the next months.

## 1.1 Outline

Chapter 2 describes the work on the realization of a tool for the application of $k$-anonymity [CDFS09] on a big data platform. The algorithm that has been selected is Mondrian, which is one of the most well known solutions for the application of $k$-anonymity. The tool implements the Mondrian algorithm within Apache Spark (http://spark.apache.org), one of the most successful frameworks for operating over large data collections. The construction of the tool had to consider several aspects that come into play when using Apache Spark, like the role of User Defined Functions (UDFs) and the careful decomposition of the anonymization task into a number of individual sub-tasks, each managed by a single node. The approach used executes a preliminary centralized phase that estimates the profile of the data distribution and then distributes the data on the computation nodes based on the results of this estimate. The use of a multiplicity of nodes guarantees high scalability. The chapter illustrates the many features of the architecture for big data used by the tools, the structure of the Mondrian algorithm, and then it illustrates the functions offered by the tools. Some preliminary considerations are reported on the future integration of this tool with the MOSAICrOWN Policy Language. The integration will eventually support the realization of additional functions in Spark.

Chapter 3 reports on the construction of a tool that verifies if a model built using machine learning allows an adversary to determine whether a specific element was present in the collection used to build the model. In general, the models built using a collection of samples may exhibit a strong affinity for each of the samples used. This can be a violation of the confidentiality of the sensitive data used in the generation of the model. Currently there is a natural expectation that a machine learning model protects the data it used for the construction, but this is not automatically guaranteed. The availability of tools like the one developed in the project will allow users of machine learning to verify whether their models leak protected information. Using the tool, the samples used for the construction of the machine learning model can be selected in order to avoid potential overfitting; alternatively, noise can be added, with approaches inspired by differential privacy, always with the objective of identifying the configuration that maximizes the quality of the model without creating undesired leakages of sensitive data.

Finally, Chapter 4 presents a few concluding remarks on the next steps that are going to occur in the realization of MOSAICrOWN sanitization tools.

# 2. Sanitization in Apache Spark

This chapter describes the tools that have been implemented to support the application of $k$-anonymity and its variants within the Apache Spark framework. The tools have been implemented starting from open-source implementations of these algorithms. The contribution is the adaptation to the Apache Spark platform and the preliminary work for their integration with the MOSAICrOWN policy language.

The chapter describes first the main technical features of the Apache Spark platform and their impact on the application of data sanitization (Section 2.1). We then describe the algorithms that have been implemented for the sanitization of data based on the $k$-anonymity paradigm (Section 2.2). We then describe the features of the implemented tools (Section 2.3) and finally provide an initial perspective on the integration of the tools with the MOSAICrOWN Policy Language (Section 2.4).

## 2.1 Overview of the Apache platform for Big Data

One of the most significant innovations in the management of big data collections is represented by the development of the MapReduce paradigm. It was clear already several years ago that the ability of a computer system to process large computations was inevitably associated with the use of parallelism and distribution. The significantly slower growth in the computational capacity of single cores, compared to what happened for more than 50 years since the development of the first computers, forces the use of a multiplicity of computational devices to operate over increasingly large data collections.

The MapReduce paradigm defines a 2-phase approach (*Map* phase followed by a *Reduce* phase) for the execution of computations over large data collections in a distributed system. Researchers in the database area have argued that the high-level design of MapReduce system corresponds to the architectures developed for the processing of queries in large distributed relational systems, where the realization of each portion of the operators in the query plan can be organized in a way that requires the decomposition of the computation across multiple nodes and the integration of the results produced by each node.

### 2.1.1 Hadoop

The crucial feature that increased the impact of the MapReduce paradigm was the availability of open-source solutions supporting the adoption of the model and the construction of applications. Among the open-source systems, the most adopted one was Apache Hadoop (http://hadoop.apache.org/), an open-source Apache Foundation software for storing and processing large data collections in distributed environments based on computer clusters. The availability of cloud systems and the development of the cloud market has immediately offered to everyone the ability to use Hadoop for the construction of applications managing large data collections. Hadoop has met with great

success and has quickly become an industrial standard, with many large-scale applications. Its open-source nature, under the Apache license, greatly facilitated its adoption.

The guiding principle of Hadoop is that of *horizontal scalability*, which assumes to increase the storage space and processing power of a hardware infrastructure by simply adding to a computer network (common workstations with standard hardware resources) new elements to distribute the workload, rather than increasing the processing capacity of a single centralized system (*vertical scalability*). In these scenarios, the computer network is commonly called a *cluster* of its node components and the processing takes place by distributing data on various nodes and performing in parallel the same operations on stored data in different nodes, according to a *shared nothing* approach, that is without sharing memory of any kind (neither primary nor secondary). In addition to the advantage of being able to adapt quickly to data of increasing size, the solution based on clusters also offers excellent failure resistance, because the data is replicated several times on different nodes of the cluster and therefore the failure of a node involves neither data loss nor interruptions or slowdowns of executions in progress. Originally, Hadoop consisted of two main components, HDFS and Hadoop MapReduce.

- **Hadoop Distributed File Sytem (HDFS)**: is a distributed file system written in Java and executable on clusters. HDFS is capable of storing large files fragmenting them on the cluster nodes, managing reliability through data replication. A Hadoop cluster contains a main node (called *NameNode*) and several controlled nodes (called *Worker* nodes). The main node manages the distribution of blocks of a file on the different controlled nodes and access to these blocks is based on the principle of moving processes on data instead of vice versa, or rather to have the processing carried out by the node itself where data is present, to minimize the transfer of data on the network. The structure of HDFS assumes that the data are organized as collections of items, each associated with a key. The distribution of the items on the storage nodes is based on the application of deterministic functions; a hash function is the default approach, but it can be redefined based on the specific application profile.

- **Hadoop MapReduce**: provides a parallel processing model, inspired by the homonymous project originally developed by Google. A MapReduce process (called job) is composed of: (a) a *Map* component, which applies to each record of the input file an operation (for example a simple selection) and generates for each record a pair "key, value", and (b) a *Reduce* component, which applies to each group of pairs generated from the Map with the same key value an aggregate function (for example a sum). The execution is governed by a process, called *job tracker*, allocated on the cluster's *NameNode*: the job tracker decomposes a job into work units (*tasks*) and assigns the tasks to the nodes on which the input file is distributed (according to the principle mentioned above), checks the termination of the tasks and reassigns to other nodes the tasks that fail.

An aspect that characterizes Hadoop is the attention dedicated to reliability. For both the computation and the storage service, there is continuous monitoring of the availability of each node and mechanisms that migrate the responsibility for the execution of each service to other nodes when nodes become unresponsive. For the storage service, the management of possible failures requires the use of redundancy in the representation of the data, and automatic mechanisms for copying data to other selected nodes when the level of redundancy of some portion of the data goes below a selected level. For computations, a monitoring is maintained by master nodes on the

progress of each node and automatically the responsibility for the management of computations is transfereed to other nodes if some nodes prove to be overloaded.

The combination of a Map operation followed by a Reduce operation can be repeated to realize more complex computations. MapReduce processing can be implemented using a variety of programming languages. In particular, the Java implementation requires to write only two classes that carry out respectively the Map and Reduce functions.

Starting with the initial release, which only included HDFS and MapReduce (and now is referred to as Hadoop 1.0), the Hadoop framework has evolved widely over the years by adding many other components to the original core (http://hadoop.apache.org). First, YARN was introduced in Hadoop 2.0, to support the effective management of system resources. In addition, components have been introduced that, on one hand, allow the development of applications using even simpler primitives and, on the other hand, implement mechanisms that support more efficient and effective processing of large amounts of data. Examples of the first type are *Pig*, *Hive* and *SparkSQL*: these systems permit to express operations on data using SQL-based languages and then automatically translate these operations into optimized cluster executions. *Apache Spark* is instead a system for the development of applications on big data alternative to MapReduce, which does not rely as much on the use of HDFS.

In addition to those named above, there are many other solutions that have extended the original core of the system; currently, the term used to refer to it is the "*Hadoop ecosystem*". Among these other solutions, we mention:

- libraries that implement machine learning algorithms in the Hadoop environment, such as MLlib (for Spark);

- tools for *data ingestion*, that is, for collecting, aggregating and moving from one system to another large amounts of data; the most significant tool in this area is *Apache Kafka* (http://kafka.apache.org/), which is a message broker, i.e., a tool capable of managing data flow queues that come asynchronously from external producers (for example from a network of sensors) and serve several consumers of this data (for example, but not necessarily, an analysis system based on Hadoop).

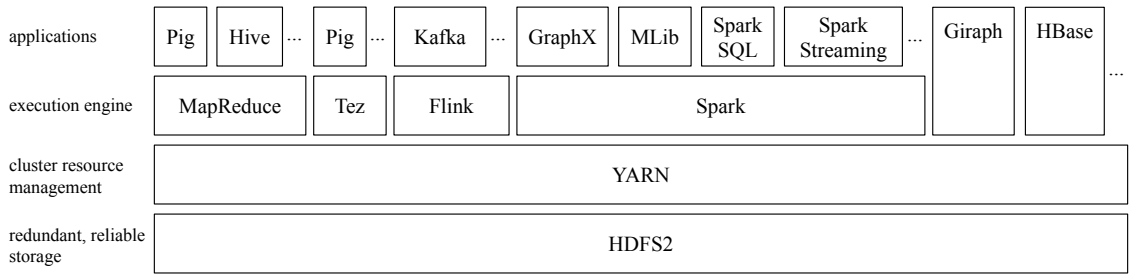


Figure 2.1: The Hadoop 2.0 architecture

Figure 2.1 provides an overview of Hadoop 2.0. The core is made up from the HDFS file system and the YARN resource manager. Above these components, among the many tools that are part of the ecosystem, by way of example in addition to those already discussed: Flink, for processing flows (streams) of data, Giraph, for processing data in the form of graphs, and HBase, which is a NoSQL system for data management

### 2.1.2 Apache Spark

The MapReduce paradigm is particularly effective for processing common analysis workloads on big data and has been very successful in this domain. However, its realization in Hadoop presents some critical issues that limit its efficiency, in particular the need to read and write the intermediate results of a job on secondary memory. In addition, the availability of only two data processing primitives (Map and Reduce, to be performed mandatorily one after the other) does not offer a direct correspondence to the structure of complex analytics. Apache Spark (http://spark.apache.org/) is an open-source project for the processing of big data that tries to overcome the limitations of MapReduce using more effectively the main memory of the nodes of a cluster, which are used for storing intermediate results. In addition, Spark has a library of predefined operators, richer than MapReduce, and thus increases the flexibility and facilitates the programming of applications. In particular, in addition to the Map and Reduce functions (and their variants) there are operations such as *filter*, *union* and *join*. Spark operates on distributed data that is typically stored on HDFS, although other distributed data storage systems may be used. A core concept at the basis of Spark is the *Resilient Distributed Dataset* (RDD), a distributed collection of objects that reside during execution typically in the main memories of the cluster nodes. RDDs can be manipulated by different operators which act in parallel and are resistant to failures thanks to replication mechanisms and, when necessary, to automatic reconstruction. Operators can be invoked interactively from console or from within a program, through APIs available for several programming languages (Scala, Python, Java, and R).

The preliminary operation consists in the launch, through the creation of a Java object, of a *SparkContext*, i.e., the process that will coordinate the execution of the Spark application on the cluster by assigning tasks to the various nodes on the basis of operators gradually invoked. In addition to the typical execution of operators, which are applied in batch mode, Spark also offers a specific tool, called Spark streaming, which supports real-time analysis on continuous data flows (streams) operating on blocks of data aggregated based on very short time intervals, called microbatches, on which to apply the same operators as for batch analysis. Finally, as previously mentioned, Spark allows, through the component SparkSQL, to express queries in SQL, which are then translated automatically into a sequence of Spark operators. For all these reasons, while the HDFS component of Hadoop is still a reference standard for storing large amounts of data in distributed environments, Spark has recently become the most used system for processing this data, even compared to MapReduce which is native in Hadoop. The investigation of how anonymization tasks can be realized in this environment is an important problem for both the research and industrial communities. The next section describes the Mondrian anonymization algorithm. In section 2.3 we describe how Spark and Mondrian are integrated.

## 2.2 Algorithms for $k$-anonymity

The privacy metrics and the classical sanitization techniques have been described in internal Workdocument W5.1 and a report on them is planned for Deliverable D5.2. The tools described in this Chapter focus on the support of $k$-anonymity. As it is common in the privacy domain, we will refer to $k$-anonymity as the property that our implementation is supporting in Spark, but we implicitly consider the refinement represented by $l$-diversity.

Several algorithms have been proposed for the application of $k$-anonymity to large data collections. The principles used for the protection had already been identified in the original paper

[Sam01]. The algorithms developed afterwards have offered improved performance, often relying on the adaptation to this domain of previous approaches for the construction of multi-dimensional indexes. The algorithm that has been selected as the basis for the implementation is *Mondrian* [LDR06].

Mondrian operates as a top-down greedy data anonymization algorithm, designed to be applied to relational datasets. Mondrian starts by mapping the domains of the quasi-identifiers attributes to generalized values (*recoding*), then creating subsets of at least $k$ elements associated with the same generalized identifier. The partitioning process can be done in four ways, which differ on the structure of the buckets, i.e., the above mentioned $n$-dimensional groups of tuples. The buckets can be associated with a range of values for each dimension that characterizes only one bucket (*strict partitioning*) or there can be an overlap between the domain covered by each bucket (*relaxed partitioning*). Strict and relaxed partitioning corresponds to the fact that buckets are either rigidly separated or overlap in their coverage of the $n$-dimensional space. In order to achieve $k$-anonymization, buckets that contain generalized tuples must not be identical on the sensitive attribute (otherwise, there would be a direct exposition of the protected value; this aspect is considered in a more careful way by the application of $l$-diversity).

If the anonymization is applied on numerical values, then each partition can be described by the numerical interval of values associated with the bucket with respect to each dimension. For textual attributes, a variety of approaches can be adopted. The most effective ones are those that start from a provided classification of the textual values, which are ordered in a way that attempts at listing near to each other terms that present a semantic correspondence. Such an input is not always available; in this case, the algorithm reverts to the use of alphabetical order.

The central step of the algorithm is the construction of the buckets in a way that respects the privacy constraints and minimizes the utility loss. The algorithm builds the buckets by evaluating all the directly available generalizations and verifying the level of utility loss, then selecting the configuration that minimizes this metric.

An aspect to consider is the number and structure of the crieria used for the partitioning of the multidimensional space. If the anonymization is done by splitting the multidimensional space on a single dimension at a time, with the impact of the split across all the tuples, then the process is *single-dimensional*. Using the single-dimensional approach, the multidimensional space is essentially organized as a rigid multidimensional grid, with regions that all refer to the same possible ranges for each dimension. Otherwise, if the anonymization process operates by splitting previously created regions on one of the available dimensions and operating the separation only a part of the regions, this leads to a more flexible structure that takes into account multiple attributes. This is the most convenient approach and is called *multi-dimensional*.

#### Single-dimensional Mondrian

Assuming there is a total order associated with the domain of each attribute in the quasi-identifier, a single-dimensional partitioning defines a set of intervals that cover the domain of each attribute. A function can map each tuple to a summary statistic for the interval in which it is contained. The result of the anonymization will be a collection of simple statistics that summarize the profile of each combination of intervals that contains (at least $k$) tuples. For instance, these summary statistics can be averages and min-max ranges.

### Multi-dimensional Mondrian

The single-dimensional partitioning model can be extended to multi-dimensional partitioning. Again, assuming a total order associated with the domain of each attribute in the quasi-identifier, a multi-dimensional partition is defined by the identification of a collection of multi-dimensional rectangular box, each associated with a range of values for each dimension. Each edge and vertex of this box may be closed or open (to denote whether the boundary value belongs to the region or not). Multi-dimensional partitioning represents the most common technique used in Mondrian, and better results can be achieved with it compared to single-dimensional partitioning.

### Strict Multi-dimensional Mondrian

A strict multi-dimensional partitioning defines a set of non-overlapping multidimensional regions that cover the domain of each quasi-identifier attribute. As before, a function can be used to map each tuple to a summary statistic for the region in which it is contained. When this function is applied to the dataset, the set of tuples in each non-empty region forms an equivalence class. The basic summary statistics is represented by the min-max range in the domain of the quasi-identifier attributes.

### Relaxed Multi-dimensional Mondrian

A relaxed multi-dimensional partitioning defines a set of potentially overlapping distinct multi-dimensional regions that cover the domain of the quasi-identifier attributes. Each tuple is then associated with every region, which works as a container for the sensitive values. A function reports summary statistics for each region that contains tuples.

| *BirthYear* | *Sex* | *ZIP Code* | *Disease* |
|---|---|---|---|
| [1960-1990] | Female | [20121-20137] | Flu |
| [1960-1990] | Female | [20121-20137] | AIDS |
| [1960-1990] | Female | 20138 | Flu |
| [1960-1990] | Female | 20138 | Hepatitis |
| [1960-1990] | Male | [20121-20137] | Flu |
| [1960-1990] | Male | [20121-20137] | Broken rib |

Figure 2.2: Single-Dimensional Mondrian anonymization, with $k = 2$

| *BirthYear* | *Sex* | *ZIP Code* | *Disease* |
|---|---|---|---|
| [1960-1980] | Female | [20121-20137] | Flu |
| [1960-1980] | Female | [20121-20137] | AIDS |
| [1970-1990] | Female | 20138 | Flu |
| [1970-1990] | Female | 20138 | Hepatitis |
| [1971-1979] | Male | 20137 | Flu |
| [1971-1979] | Male | 20137 | Broken rib |

Figure 2.3: Multi-dimensional Mondrian anonymization, with $k = 2$

**An example of Mondrian execution**

We present an extremely compact example of the application of Mondrian, in order to clarify the variants of this algorithm. Tables in Figure 2.2 and Figure 2.3 both contain 6 records, on which thet two main variants of Mondrian have been applied. Specifically, records in Figure 2.2 are the results of single-dimensional Mondrian anonymization, while records in Figure 2.3 are the results of a multi-dimensional Mondrian anonymization process. Attributes "BirthYear", "Sex" and "Zip Code" represent the quasi-identifiers in the dataset, whereas "Disease" is the sensitive attribute. Both tables represent a 2-anonymized dataset. In the single-dimensional approach, for each of the quasi-identifier attributes the domain is split in a single collection of ranges (e.g., [20121-20137] and [20138-20138], compacted as a single value, for the ZIP code). In the multi-dimensional anonymization, the attributes are not split for all tuples in the same set of ranges (e.g., we notice that we have three overlapping ranges [20121-20137], [20137-20137] and [20138-20138] for the ZIP code). In both anonymizations, the number of tuples presenting each combination of the values of the three quasi-identifiers is equal to 2.

## 2.3 Features of the anonymization tools

The implementation of the Mondrian anonymization algorithm described in this chapter is available at https://github.com/mosaicrown. The software was created starting from an open source implementation. The plan for this activity is to release an open source system and starting from an existing one was both reducing the time required to run experiments and was also seen as a way to support the open source principles shared by academic partners. The amount of work dedicated to the extension of the tools has been extensive and it can be estimated that a similar effort would have been spent if the development had started from scratch. The effort has focused on improving the flexibility of the software, its integration with Apache Spark, and organizing the system in a way that will support the integration tool with the MOSAICrOWN Policy Language.

The first important feature of the anonymization tool is the support for the execution of the algorithm on multiple nodes. The original code was only executable in standalone mode on a single node. The crucial aspect to consider in the adaptation to the distributed setting is the identification of a correct partitioning strategy for the data. A simple approach where the data is distributed randomly on the nodes, and then each node is responsible for the creation of an anonymous representation of the data locally available on the node, is going to produce in most cases anonymizations with worse utility than the one obtained by identifying a preliminary distribution strategy. Performance and scalability would be optimal, but the quality of the queries executed on the anonymized data is lower.

The pseudocode of the Mondrian algorithm implemented by the tool is presented in Figure 2.4. The first step consists in the identification of the criteria to use to split the database on the distinct nodes involved in the computation. Function *chooseDimension* is the function used in Mondrian to determine the dimension to use for splitting at each step a partition into two parts. The function is applied as a first step to determine the dimension to use to split the data on the nodes of the Spark cluster (procedure *sparkSplitOnDim*(*DB*)). Then, the Mondrian partitioning method (*Anonymize*) is applied by each node on its partition. The structure of the *Anonymize* method is the one characterizing the original presentation of the algorithm [LDR06].

The function *chooseDimension* has been adapted to support the evaluation of the partitioning strategy. The function relies on an estimate of the quality obtained by using a given dimension as

```
firstDim = chooseDimension(DB)
sparkSplitOnDim(DB,n)
spark.Anonymize(partition)

Anonymize(partition)
  if (no allowable multidimensional cut for partition)
    return φ: partition → summary
  else
    dim ← chooseDimension(partition)
    fs ← frequencySet(partition,dim)
    splitVal ← findMedian(fs)
    lhs ← { t ∈ partition : t.dim ≤ splitVal }
    rhs ← { t ∈ partition : t.dim > splitVal }
    return Anonymize(rhs) ∪ Anonymize(lhs)
```

Figure 2.4: Pseudocode of the Mondrian algorithm adapted to Spark

criterion for the distribution of the data. The dimension that produces the partitioning with the best quality will be used to partition the data to the distinct nodes. The current implementation is going to be extended in the next period, with the implementation of a sampling approach, to produce in a more efficient way the evaluation of the quality deriving from the use of a given dimension. For data where it is known which is the criteria to use for the distribution of the data to the nodes, this input can be provided directly to the tool, accelerating the computation.

Attention was also dedicated to the improvement of flexibility. The starting code base was quite rigid in the format of the input, managing only numerical attributes and specific positions for quasi-identifiers and sensitive attributes. A natural requirement for a flexible sanitization tool to be used in MOSAICrOWN is the ability to be applied on arbitrary data collections, with both numerical and textual domains. The support for extensive input parametrization was then introduced.

The tool relies on a Python program for the activation and the management of the anonymization. The Python program invokes a tool that is written in Scala and realizes the core of the anonymization, running on the Spark platform. The construction of the buckets is based on the recursive evaluation of the *chooseDimension* function, which evaluates the *Normalized Certainty Penalty* (NCP), a measure of the information loss produced by the anonymization. Alternative functions for the estimation of the quality of the buckets are being evaluated. Another collection of functions apply basic protection to the data by dropping attributes that must be omitted as they represent direct identifiers. This task is managed with the use of newly specified Spark UDFs. These UDFs operate directly on the structure of tuples.

The tools offer the opportunity to read and write the data directly on files stored in a CSV format or from a HDFS structure. When using HDFS, a function has been built that permits as a final step to extract the data that was written by the nodes onto the HDFS structure and produces a CSV representation of the data. The work on these transformations has demonstrated that there is some opportunity for improvement in the maturity of Apache Spark, with errors in the execution that have been solved by upgrading to the most recent version. Other small anomalies have been observed in the implementation of the system and workarounds to anomalies have been

implemented to support the smooth execution of the transformations.

Another function that was implemented is the support for the execution of generalization using a provided hierarchy. The hierarchy for the values of an attribute is described in a dedicated file provided as input. This hierarchy is then considered for the anonymization executed within Spark. This step increases the flexibility of the anonymization, at the cost of executing an ad-hoc definition of the hierarchical structure adequate for a specific domain. For many applications, it can be envisioned that predefined hierarchies are available and can be provided as an input. The automatic generation of such hierarchical structures is not expected to be part of the investigation within MOSAICrOWN.

Support is currently at a preliminary stage for the specification of $l$-diversity. The construction of each bucket verifies that the sensitive values are not all identical, but the application of a threshold on the presence of at least $l$ distinct values for the sensitive attribute is only applied in a dedicated implementation of the function. The line of research that is planned to be investigated looks at alternative measures of diversity on the protected values. In general, there is a delicate balance between the management of privacy and utility, and the rigid specification of a numerical threshold is not necessarily the best option for each application domain. The consideration of $t$-closeness, based on the evaluation of metrics on the distribution of the sensitive values for each bucket, offers a more precise characterization and can provide benefits in some applications. The integration of these metrics with the features of a big data platforms like Spark will be the subject of dedicated work.

As part of an independent line of development, other anonymization algorithms have been implemented in a preliminary way as alternatives to the use of Mondrian. Specifically, a variant of the Incognito algorithm [LDR05] has been realized. This is built in Scala and its implementation has been tested. The goal is to consider the realization of alternative solutions for $k$-anonymizations, offering to the user of the tool chain several options for the processing of a given data collection. Mondrian is typically considered the reference solution for the application of $k$-anonymization and this justifies its selection as the main algorithm, but in this domain there is no one-size-fits-all solution and the availability of multiple options can prove to be the best approach for a concrete system.

### Experiments

The behavior of the implemented tools has been evaluated in a series of experiments. The main goal of the experiments was the verification that the use of Spark was able to improve the performance in the execution of the anonymization. The Health Dataset [Dep] was used, which contains 163,000 records with 12 attributes. Among the 12 attributes, 2 attributes were filtered and 7 attributes were classified as quasi-identifiers. The experiments for a single client were run a notebook PC (Lenovo Thinkpad T410, Intel i5, 8GB of RAM), the experiments on the cluster were run on three machines connected on a LAN ((1) the notebook PC above, (2) a Lenovo Thinkpad P1, Intel Xeon, 64GB, (3) a desktop PC with Intel i7, 16 GB). The results of the execution of the Mondrian algorithm, with the relaxed approach, are reported in Figure 2.5. The experiments confirm that the use of Spark leads to a significant reduction in the execution time. The complexity of the anonymization decreases as $k$ grows.

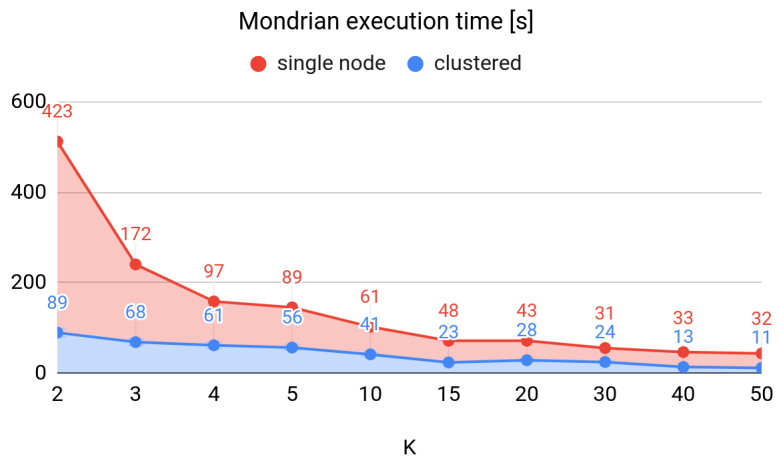


Figure 2.5: Results of the execution of the algorithm

## 2.4 Integration of anonymization tools with the MOSAICrOWN Policy Language

The implementation of the anonymization algorithms considers two goals. The first goal is the investigation of techniques for the efficient application of anonymization over large data collections. The second goal is the construction of an environment for digital data markets where there is a collection of data protection tools, implementing a variety of approaches and techniques, and the application of the techniques is driven by a high-level specification, expressed using a flexible policy language. This would allow users of the digital data market to (semi-)automatically apply transformations and purpose restrictions specified by the policy. This would give a guarantee that the data processing services offered by the market are consistent with the preferences expressed by data subjects and by the upstream data controllers.

The work on the definition of the policy language is still ongoing and the first version will be presented in Deliverable D3.3, due at M18. This section discusses the features that are expected to be available in the language to support the representation of the anonymization requirements and its (semi-)automatic application. The general framework supporting the integration is described in Deliverable D3.2 “Preliminary version of tools for the governance framework”.

The central property of the MOSAICrOWN Policy Language is the ability to define authorizations for the parties accessing the system. Compared to traditional access/usage control languages, the structure is richer, as it supports the explicit representation of the purpose. Also, the policy model supports the representation of transformations on the data, with the goal to deliver sophisticated reasoning solutions. The availability of such a model represents a significant contribution to the realization of a richer environment for the management of security and privacy requirements, particularly critical when considering the construction of large digital data markets.

We consider here a model that could become part of the MOSAICrOWN Policy Language. It is to note that the language will specify other aspects associated with sanitization, like restrictions on attribute visibility or parameters for the application of differential privacy. We present the definition of the profile of an anonymization of the data based on the definition of identifiers to drop and quasi-identifiers to suppress/generalize in order to protect the association between the sensitive information and the identity of users. The model specifies the transformation to

anonymize the data assuming that the function is implemented by the data management platform. The transformation specifies the output format, which for anonymization is defined as a subset of the attributes appearing in the schema of the input. The attributes that are part of the schema can be quasi-identifiers or sensitive attributes. There is a dual/opposite nature for quasi-identifers and sensitive attributes. For quasi-identifiers, the tuples must be grouped so that each bucket is associated with the same collection of values for quasi-identifiers. Whereas, for sensitive attributes it is crucial to have a variety of values. Then, for the application of anonymization, value $k$ determines the minimum size of each bucket of tuples, i.e., the group of tuples with the same quasi-identifiers. To specify that the sensitive attributes, belonging to the elements of a bucket, must not all be identical, but have a minimum number of distinct values, an additional $l$ parameter (less than $k$) can be specified, which determines the minimum number of distinct values that should appear in each bucket.

The format proposed for the representation of these properties is JSON-LD, described in Deliverable D3.2. An example of specification is in Figure 2.6.

The output of the transformation is then an object that can be the target of authorizations. Parties with the privilege to view all the inputs are implicitly authorized to view the result of the anonymization.

```
{
    "@context": [
        "http://www.w3.org/ns/odrl.jsonld",
        "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
    ],
    "@type": "Set",
    "permission": [{
        "assignee": "http://org/user",
        "target": "http://example.com/Customer",
        "action": [{
            "rdf:value": { "@id": "odrl:anonymize" },
            "refinement": [
                {
                    "leftOperand": "mosaicrown:method",
                    "operator": "eq",
                    "rightOperand": { "@value": "l-diversity",
                                      "@type": "xsd:string" }
                },
                {
                    "leftOperand": "mosaicrown:k",
                    "operator": "eq",
                    "rightOperand": { "@value": "10",
                                      "@type": "xsd:integer" }
                },
                {
                    "leftOperand": "mosaicrown:l",
                    "operator": "eq",
                    "rightOperand": { "@value": "3",
                                      "@type": "xsd:integer" }
                },
                {
                    "leftOperand": "mosaicrown:identifier",
                    "operator": "isAllOf",
                    "rightOperand": [
                        "http://example.com/Customer/CustomerID",
                        "http://example.com/Customer/FirstName",
                        "http://example.com/Customer/LastName"
                    ]
                },
                {
                    "leftOperand": "mosaicrown:quasi-identifier",
                    "operator": "isAllOf",
                    "rightOperand": [
                        "http://example.com/Customer/ZIP",
                        "http://example.com/Customer/BirthDate",
                        "http://example.com/Customer/TransactionTime"
                    ]
                },
                {
                    "leftOperand": "mosaicrown:sensitive",
                    "operator": "isAllOf",
                    "rightOperand": [
                        "http://example.com/Customer/Amount",
                        "http://example.com/Customer/StoreCategory"
                    ]
                }
            ]
        }],
        "output": "http://example/anonCustomer",
        "purpose": "marketing"
    }]
}
```

Figure 2.6: An example of policy declaration for an anonymization action.

# 3. Support for Differential Privacy

In this chapter, we describe a solution for monitoring privacy risk and data leakage for differential privacy, a perturbation-based (i.e., randomized) anonymization technique. The solution permits to measure the vulnerability of training records in a data set after releasing an inference model being optimized on them. Vulnerability is defined as the precision of an attacker to sense the presence of an individual record. Hence, this thread model is known as membership inference attacks. This thread model provides not only a way to monitor privacy risks but also a privacy metric to investigate data sanitization techniques for machine learning – as will be detailed in D5.2. In the first phase of MOSAICrOWN we investigate potential ways to improve such data sanitization techniques to provide differential privacy, e.g., by analyzing membership inference attacks.

The membership inference attack, abbreviated as MIA, is realized as a programming library to simulate such attacks on machine learning (ML) models. The MIA library integrates in development processes for model training and deployment, but may also be provided as independent microservice for ML quality assurance. In the following, the inference model to be evaluated is considered a deep neural network architecture.

In Section 3.1 we briefly define differential privacy and in Section 3.2, membership inference attacks and related terminology are explained. Afterwards, their implementation in the MIA library is detailed in Section 3.3.

## 3.1 Differential Privacy

In contrast to anonymization methods based on generalization, *differential privacy* (DP) [Dwo06] anonymizes a dataset $D = \{d_1, \ldots, d_n\}$ by perturbation. Informally, *mechanism,* within the context of differential privacy, refers to a randomized algorithm. Such mechanisms perturb either the data values $d \in D$, called *local DP*, or the function result $f(D)$, called *central DP*, via additive noise fine-tuned to the desired function (to be evaluated on the data) and the privacy parameter $\varepsilon$. In the following, we give formal definitions.

### 3.1.1 Central DP

In the central model the aggregation function $f(\cdot)$ is evaluated and perturbed by a trusted server. Due to perturbation it is no longer possible for an adversary to confidently determine whether $f(\cdot)$ was evaluated on $D$, or some neighboring dataset $D'$ differing in one element. Thus, assuming that every participant is represented by one element, privacy is provided to participants in $D$ as their impact of presence (absence) on $f(\cdot)$ is limited. *Mechanisms M* fulfilling Definition 1 are used for perturbation of $f(\cdot)$ [DKM$^+$06]. We refer to the application of a mechanism $M$ to a function $f(\cdot)$ as *central differential privacy*.

**Definition 1 ($(\varepsilon,\delta)$-central differential privacy)** *A mechanism $M$ gives $(\varepsilon,\delta)$-central differential privacy if $D, D' \subseteq DOM$ differing in at most one element, and all outputs $S \subseteq R$*

$$\Pr[M(D) \in S] \leq e^{\varepsilon} \cdot \Pr[M(D') \in S] + \delta$$

Note that $\delta$ is a (negligible) value allowing DP to be violated with very small probability leading to more efficient or accurate mechanisms and is commonly used in the context of machine learning. CDP holds for all possible differences $\|f(D) - f(D')\|_2$ by adapting to the global sensitivity of $f(\cdot)$ per Definition 2, which is the maximum influence (absolute difference) a single change can have on the function evaluation.

**Definition 2 (Global $\ell_2$ Sensitivity)** *Let $D$ and $D'$ be neighboring. The global $\ell_2$ sensitivity of a function $f$, denoted by $\Delta f$, is defined as*

$$\Delta f = max_{D,D'} \|f(D) - f(D')\|_2.$$

### 3.1.2 Local DP

We refer to the perturbation of entries $d \in D$ as local differential privacy [WBLJ17]. LDP is the standard choice when the server that evaluates a function $f(D)$ is untrusted. We adapt the definitions of Kasiviswanathan et al. [KLN+08] to achieve LDP by using local randomizers $LR$.

**Definition 3 (Local differential privacy)** *A local randomizer (mechanism) $LR : DOM \to S$ is $\varepsilon$-local differentially private, if $\varepsilon \geq 0$ and for all possible inputs $v, v' \in DOM$ and all possible outcomes $s \in S$ of $LR$*

$$\Pr[LR(v) = s] \leq e^{\varepsilon} \cdot \Pr[LR(v') = s]$$

$\varepsilon$-local algorithms are $\varepsilon$-local differentially private [KLN+08], where $\varepsilon$ is a summation of all composed local randomizer guarantees.

**Definition 4 (Local Algorithm)** *An algorithm is $\varepsilon$-local if it accesses the database $D$ via $LR$ with the following restriction: for all $i \in \{1,\ldots,|D|\}$, if $LR_1(i),\ldots,LR_k(i)$ are the algorithms invocations of $LR$ on index $i$, where each $LR_j$ is an $\varepsilon_j$-local randomizer, then $\varepsilon_1 + \ldots + \varepsilon_k \leq \varepsilon$.*

## 3.2 Membership Inference Attacks

The goal of a membership inference attacker is to identify records that have been used to train an inference model, i.e., an attacker aims to disclose the fact that a record belonging to an individual is part of the training set. Revealing this is called *membership inference*. The simplicity of the attack (inferring a binary information) makes it possible to illustrate how much privacy is compromised by publishing a model. If an attacker can confidently and successfully infer membership of a data record, information about the record and its corresponding person is leaked through the model. Yet, the attack does not quantify this. It only indicates the leakage of private information, as the attack would not be possible otherwise. Even though membership inference attacks are also

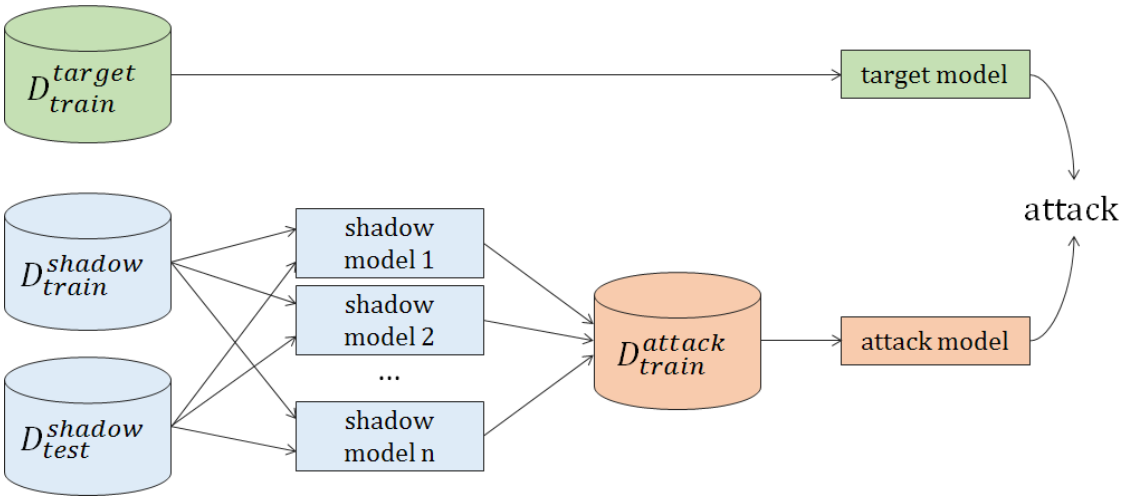


Figure 3.1: Process of membership inference attacks

applicable to other statistical methods and queries in general, our focus is on machine learning applications only. While there are multiple membership inference attacks published in the literature [YGFJ18], [NSH18], we focus explicitly on the work by Shokri et al. [SSS16]. Their idea is to exploit the fact that with a limited set of training data a classifier is unlikely to generalize perfectly, especially for complex tasks such as image classification. Hence, a classifier remembers information about its training data being known as overfitting. This affects the confidence of a model when classifying a record from the training set and a record not seen throughout the training process. These differences of confidence are learned by the attacker to predict if a record has been in the training set.

The model (classifier) to be attacked is called *target model*. In general, membership inference attacks are model-agnostic. The only requirement is that the outputs of the target model must contain confidence values for each class as provided by naive bayes models, logistic regression or softmax layer of neural networks. An attacker creates multiple copies of the target model that are called *shadow models*. All shadow models are trained the same way as the target model is trained. However, the attacker does not know about the training set of the target model which is why they have or create a custom training data set for each shadow model. Hence, we assume the attacker to know the target model architecture or to have access to the same ML service used for training the target model. Also, expect the attacker to know about the hyper parameters used to train the target model. In the case of using a ML web service, we expect the service to use the same or a very similar setting. The attacker utilizes their shadow models to mimic the behavior of the target model, but with the additional information of whether an instance is part of the training set. This gives them the opportunity to train the final *attack model* to learn the differences in confidence using outputs of all shadow models. The process is illustrated in Figure 3.1.

As membership inference attacks are model agnostic, there are just a few requirements towards the target model. The target model (as well as a shadow model) describes a function $f_{target} : \mathscr{X} \rightarrow [0,1]^C$ mapping from a feature space $\mathscr{X}$ to a probability output space $[0,1]^C$ where $C$ denotes the number of classes the target model is supposed to distinguish. A vector $v \in [0,1]^C$ has the property $\sum_i v_i = 1$, thus each vector element $v_i$ represents the confidence of the model or its posterior probability $P(c|X)$ for class $c$ given some input $X \in \mathscr{X}$. Typically, the predicted class is derived by choosing the class with the highest confidence value, i. e. $\hat{c} = \arg\max_i f_{target}(X)$.

To train the target model a private training set $D_{train}^{target}$ is used. We define a training set to contain entries $(X, y)$ with $X$ being the feature of one instance and $y$ the class label. An attacker might have access to a record $X$ that is in $D_{train}^{target}$ but they are not aware of the fact that $(X, \cdot) \in D_{train}^{target}$.

In contrast, for each shadow model $f_{shadow_i}$ the membership information of records in the training set $D_{shadow_i}^{train}$ and test set $D_{shadow_i}^{test}$ are known to the attacker. We denote the number of shadow models by $N_{shadow}$. Typically, a pool of shadow training data $D_{shadow}^{train}$ is used to create the training set $D_{shadow_i}^{train} \subset D_{shadow}^{train}$ for every shadow model. Thus, training sets of different shadow models may overlap. The same applies for the test sets. However, for all $D_{shadow_i}^{train} \cap D_{train}^{target} = \emptyset$ and $| D_{shadow_i}^{train} | = | D_{train}^{target} |$ is required. This guarantees that shadow models have a similar data foundation as the target model. Hence, it is assumed that in every shadow training set the distribution of classes is similar to $D_{train}^{target}$. Also, the same hyper parameters must be used in the training process. This is simple if the model architecture and training parameters are known. If a ML service, such as [Goo], is called for training, it becomes more difficult. The easiest solution from an attacker's perspective is referring to the same ML service.

After training all shadow models, the attacker has multiple models they expect to behave similarly to the target model. Based on them, she trains an attack model. Referring to [SSS16], an attack model $f_{attack_c} : [0,1]^C \to \{0,1\}$ is trained individually for every class $c$. In the end, the attacker has $C$ attack models, one for each class. The inputs of an attack model are the confidence values of the target model and the outputs are 1 for membership and 0 for non-membership. Given some instance X that is classified as class $\hat{c}$ by the target model, the attacker computes

$$member = f_{attack_{\hat{c}}}(f_{target}(X)) \tag{3.1}$$

To train each attack model, a training set is constructed using shadow models. Given class $c$, and shadow model $i$, we compute a fraction of the final set as follows.

$$D_{train}^{attack_{ci}} = \{(f_{shadow_i}(X), 1) \mid (X,y) \in D_{train}^{shadow_i} \wedge y = c\} \tag{3.2}$$

$$\cup \{(f_{shadow_i}(X), 0) \mid (X,y) \in D_{test}^{shadow_i} \wedge y = c\} \tag{3.3}$$

Here, all predictions on the train set are labels as 1 and on the test set as 0. In addition, only instances $X$ are considered where the corresponding class label $y$ is equal to $c$. The complete training set of the attack model is constructed by joining all fractional parts

$$D_{train}^{attack_c} = \bigcup_{i=1}^{N_{shadows}} D_{train}^{attack_{ci}} \tag{3.4}$$

A graphical illustration of this process is depicted in Figure 3.2. Also, note that every $D_{train}^{attack_c}$ is supposed to be balanced, i. e. it contains the same number of entries labeled with 0 ("out") and 1 ("in"). There are no assumptions about the type of attack model to be used. Generally speaking, any binary classifier may be used. Nonetheless, the MIA library focuses on using fully-connected neural network model architectures as attackers, compare with [SSS16]. In the end, necessarily, the attacker needs to be evaluated on the target model in order to measure its privacy leakage. Therefore, another data set $D_{test}^{attack_c}$ is created which contains an equal number of instances from the target model's training set and an independent test set.

$$D_{test}^{attack_c} = \{(f_{target}(X), 1) \mid (X,y) \in D_{train}^{target} \wedge y = c\} \tag{3.5}$$

$$\cup \{(f_{target}(X), 0) \mid (X,y) \in D_{test}^{target} \wedge y = c\} \tag{3.6}$$

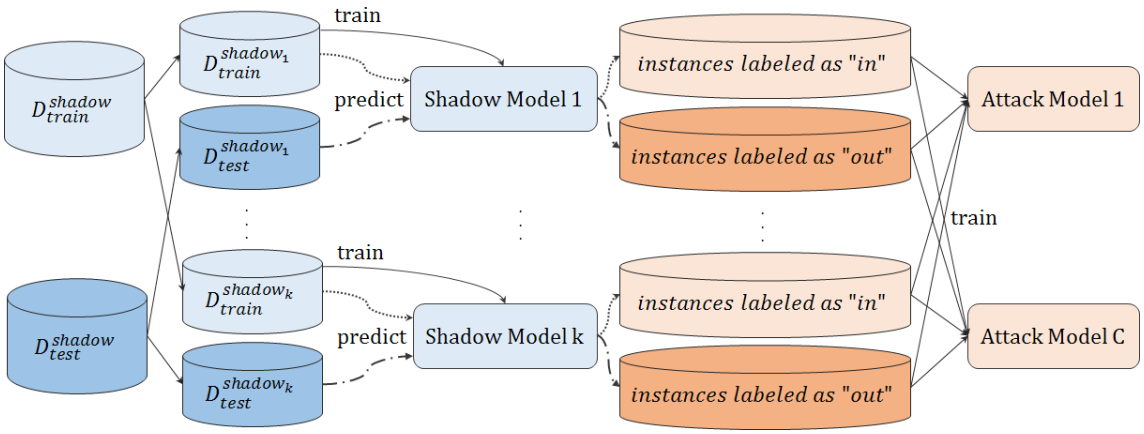


Figure 3.2: First, train and test data sets for each shadow model are drawn from a pool of train and test data. Second, the shadow models are trained with the respective data. Third, the confidence values when predicting train and test data form the attack train set together with "in" and "out" labels.

The attack is regarded successful if most members are correctly identified. Thus, the attack models are evaluated using precision $prec = \frac{TP}{TP+FP}$ and recall $rec = \frac{TP}{TP+FN}$ scores. The true positives and true negatives indicate correctly classified members and non-members, respectively. In consequence, the precision scores evaluate how often the identification of members has been correct and the recall score shows how many members have been found.

## 3.3 Library Design

We realize this attack in the MIA library with only a few classes. The architecture of the library and how a data analyst might use it as an auditing tool is illustrated in Figure 3.3.

Everything happens inside an instance of the core class `MIApplication`. It is responsible for coordinating all parts of a membership inference attack which includes splitting the data for target model and shadow model training. When utilizing MIA as a library or service, the data analyst inputs a data source, where $D_{train}, D_{test}$ can be obtained from, and a model architecture plus its (training) hyper parameters. To be as flexible as possible, data is accessed through a `DataSet` object which enables the library to use any kind of source, e. g. files, database servers, etc. The model architecture gets instantiated multiple times: once as `TargetModel` and $n$ times as `ShadowModel`. First, the target model is trained according to its hyper parameters and configuration input from the data analyst. Second, the shadow models are trained. In fact, they are organized within a `ShadowGroup` which is a convenience wrapper to execute an operation with all shadow models at once, i. e. without the need to iterate over them. The `ShadowGroup` serves as source for generating $D_{train}^{attack_c}$ by an instance of class `AttackTrainSetGenerator`. Each one of the $C$ `AttackModels` contained in the `Attacker` is trained on its respective data set generated this way. Afterwards, the `TargetModel` is evaluated on its training and testing sets $D_{train}^{target}, D_{test}^{target}$ inside the `AttackTestSetGenerator` to construct $D_{test}^{attack_c}$. Finally, the precision and recall of every `AttackModel` is estimated and reported by the MIA library as quality certificate for the target model. The MIA library is implemented in pure Python and can be bundled as a single Python package. All models (neural networks), their training and evaluation process are realized via differ-

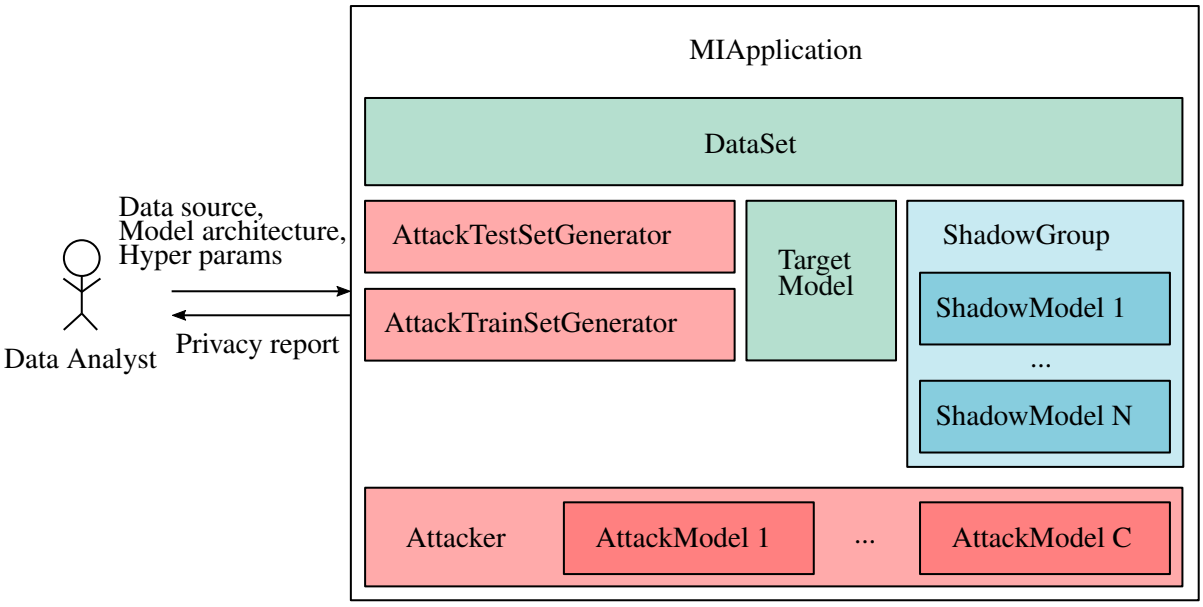


Figure 3.3: Architecture of the MIA library

entiable data processing libraries[1]. In particular, MIA builds on top of the Tensorflow [AAB+15] Python framework. Utilizing privacy-preserving optimization techniques when training the target and shadow models has demonstrated to reduce the privacy loss and yielded favorable quality results. Hence, MIA allows to use such techniques.

## 3.4 MIA Service

The functionality of MIA can be wrapped and exposed as a browser-consumable service. Hence, the data analyst uses a web user interface instead of interacting through code with the library. Nonetheless, the inputs remain the same but are uploaded to the server before running the quality audit of MIA. The outputs are also identical but might be presented more user-friendly. An overview of the service structure, which could also be extended for integration with the Apache Spark environment, is displayed in Figure 3.4.

The client (data analyst) performs an initial request to the service's host which is processed by the UI service run inside a Python Tornado HTTP server process. The UI service responds by transferring the user interface (UI files) to the client's browser which renders the client application. The user interface of the client application is realized with SAPUI5, a design framework that uses JavaScript and a model-view-controller architecture. The UI contains input fields where the analyst can upload new data sets, model architectures and hyper parameter configuration. Further, she can start, stop and monitor current audits. When uploading or performing an action on a MIA audit, the UI directs the browser to a dedicated MIA REST service wrapping the functionality of the MIA library. Uploaded data sets are stored as files, while configurations, models and their parameters are stored within a MongoDB data base system. The MongoDB also contains meta information about the data set, e.g., the path of their files and train-test-splits. To start a MIA audit execution, the client selects an uploaded data set and a model. This triggers the MIA service to

[1]Differentiable programming is a programming paradigm for automatic differentiation as used to solve gradient-based optimization problems, e.g., gradient descent.

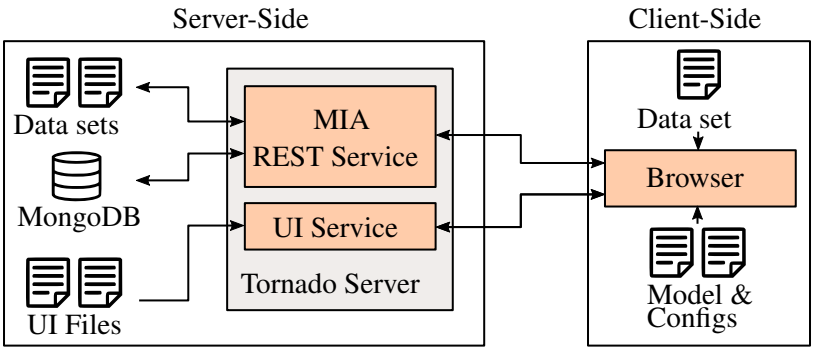


Figure 3.4: MIA Service

spawn a new process. This process behaves as a local execution of the MIA library. It is given the required input information stored in the MongoDB. The browser queries the status of a run in a fixed interval to monitor the status of an audit. It allows to start multiple audits in parallel.

# 4. Conclusions

The tools described in this deliverable are representative of the effort currently ongoing in MOSAICrOWN toward the realization of a modern digital data market, where the benefits deriving from access to a multiplicity of data sources are obtained without violating the confidentiality and integrity requirements associated with the data. An important example of these benefits is represented by the recent COVID-19 pandemic, which has demonstrated the central role that access to detailed data can have in the definition of medical, social and economic policies, particularly for democratic societies, where citizenship demands visibility of the sources used to enact restrictions of personal freedom. The benefit has been significant, but it could have been greater if privacy concerns had been adequately supported by dedicated technical solutions applicable with limited effort. In the same way as better scientific knowledge on virology and genomics will give us access to faster and better tools to face future threats, the availability of improved solutions for the privacy-compliant management of data analysis will allow us to be better prepared and more quickly and more effectively obtain the benefits that the analysis of data can provide.

The work in the project will continue to develop improvements to the tools described in this document and to other new tools. The functions offered will be extended and the flexibility in the application of the tools will increase, facilitating their adoption in applications. Another central line of development will focus on the integration with the MOSAICrOWN Policy Language. The constructs to use for the representation of the sanitization requirements will be finalized and software modules will be built that will support the verification of properties on sanitization and the mapping between the specified policy and its implementation in the tools. The work will also consider several approaches for sanitization. In addition to the Mondrian algorithm for $k$-anonymity, other algorithms will be developed and the integration with the wrapping tools and query optimizers developed in Work Package 4 will be explored.

Support will also be offered, both in the policy language constructs and in terms of sanitization tools, for the application of differential privacy, which is going to be a central aspect of the work done within Use Case 3. It is to note that both approaches have domains where they can be the best option. Recently, a lot of attention has been dedicated to differential privacy, but this interest does not reduce the impact that approaches based on $k$-anonymity have on the construction of real applications. An example of the role that both approaches can play is represented by the current experience of the US Census Bureau, which is applying both approaches in the release of its statistical data. The approach considered by MOSAICrOWN, which looks at both families of techniques, is then aligned with the state of the art in the design and implementation of sanitization techniques for large data collections.

# Bibliography


[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[CDFS09] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. Theory of privacy and anonymity. In M. Atallah and M. Blanton, editors, *Algorithms and Theory of Computation Handbook (2nd edition)*. CRC Press, 2009.

[Dep] Department of Health of the United States. Official hospital patients data. https://www.data.gov/health/.

[DKM+06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our Data, Ourselves. In *Proc. of Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2006.

[Dwo06] Cynthia Dwork. Differential Privacy. In *Proc. of Colloquium on Automata, Languages and Programming (ICALP)*, 2006.

[Goo] Google. Google Cloud AI. https://cloud.google.com/ml-engine/.

[KLN+08] Shiva P. Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 2008.

[LDR05] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 49–60. ACM, 2005.

[LDR06] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Mondrian: Multidimensional k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 25. IEEE Computer Society, 2006.

[NSH18] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks. *arXiv preprint arXiv:1812.00910*, 2018.

[Sam01] Pierangela Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(6):1010–1027, November/December 2001.

[SSS16] Reza Shokri, Marco Stronati, and Vitaly Shmatikov. Membership inference attacks against machine learning models. *CoRR*, 2016.

[WBLJ17] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. Locally differentially private protocols for frequency estimation. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 729–745. USENIX Association, 2017.

[YGFJ18] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 268–282. IEEE, 2018.